Kafka源码深度剖析-第一天

一 为什么有会Kafka源码课?

1. 在大数据的场景里面如何应对数据激增，数据复杂度增加以及数据变化速率变快，这都是体现了大数据架构师，Java架构师的功力。而Kafka能很好的解决这些问题。
2. Kafka的源码是众多开源的技术里面代码质量比较高的一个，所以本身研究它的源码就比较有观赏性。

3. 在众多大数据技术里面，Kafka是难度较大的一个技术。

4. 价格是价值的货币表现形式。

二 阅读Kafka源码需要储备什么知识?

1. Java IO，多线程知识，Java并发
2. 一点Scala
3. Kafka基础知识

三 源码导入流程

02 图解Kafka源码-源码阅读准备之源码环境准备.pdf

四 本次训练营收获

道的层面：

（1）分布式系统源码阅读方式

（2）分布式消息系统的架构设计原理（高并发，高可用，高性能）

术的层面：

（1）Kafka客户端发送消息的核心流程

（2）Kafka客户端内存池的架构设计

（3）Kafka客户端的容错设计

（4）Kafka支持超高并发的网络架构设计

（5）Kafka高性能，高并发，高可用的架构设计原理

（6）Kafka副本同步机制原理

（7）Kafka元数据核心管理流程

五 本次源码讲解的方式

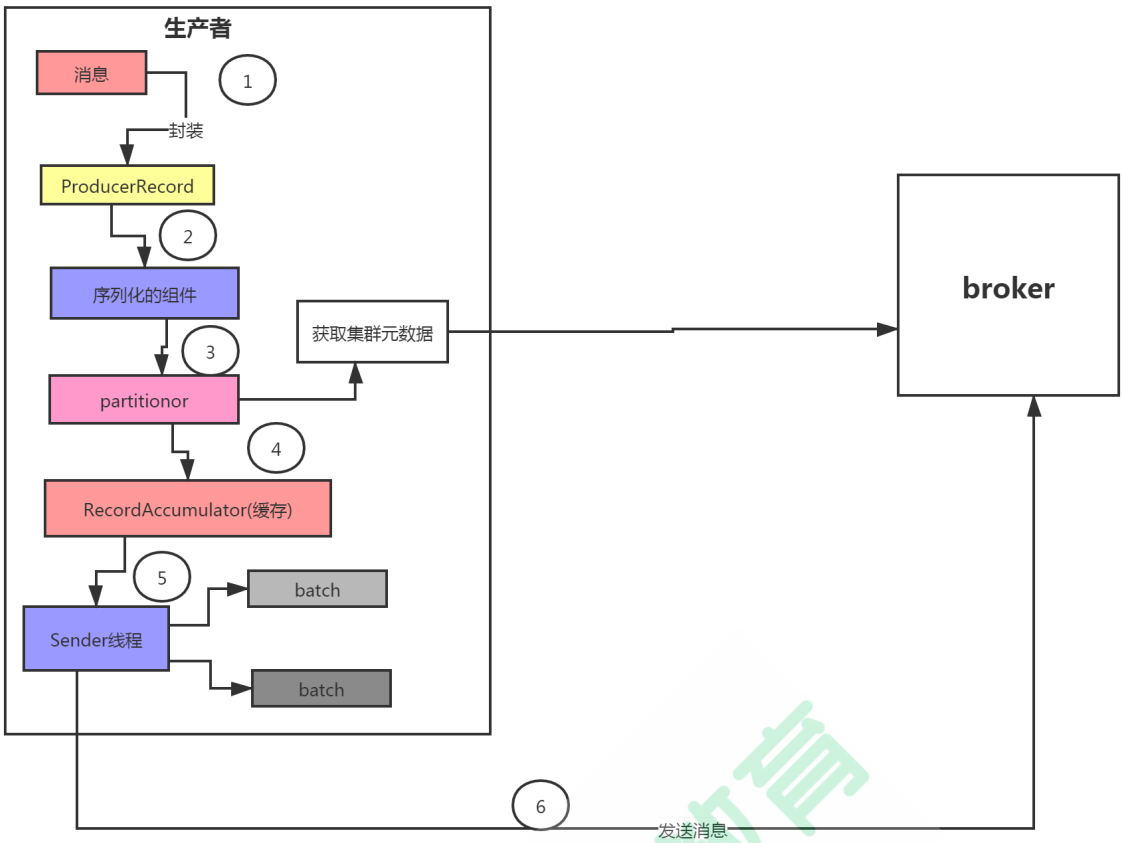
1. 场景驱动
2. 图解源码

六 本次课知识点

场景驱动方式：

（1）生产者是如何发送数据的？ 技术含量很高

6.1 生产者发送消息流程回顾



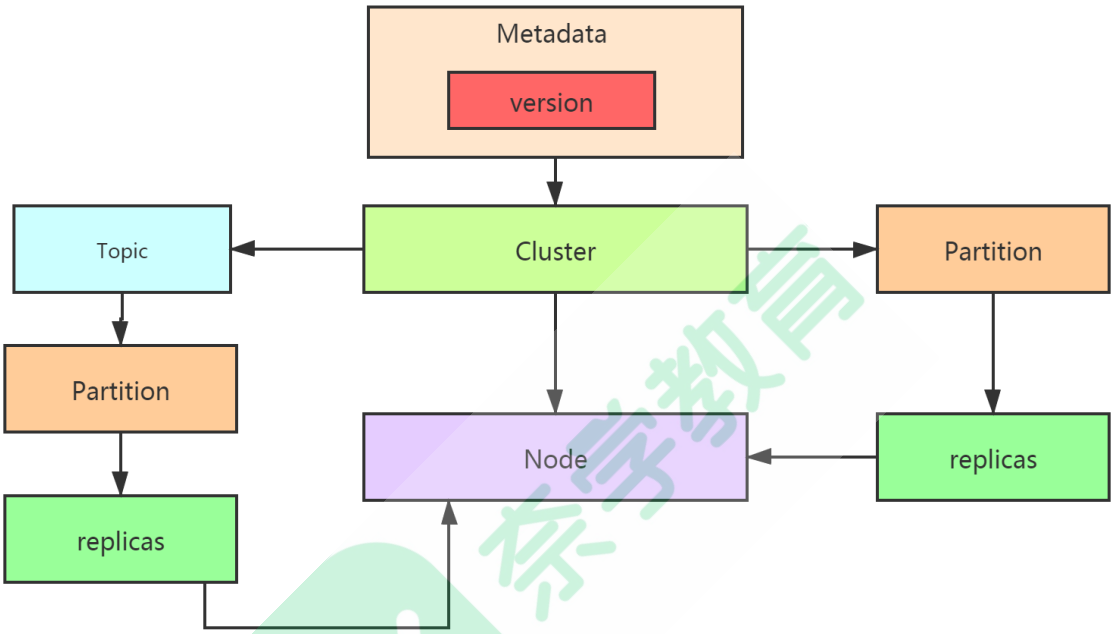
6.2 从demo 入手

```
public class Producer extends Thread {
    private final KafkaProducer<Integer, String> producer;
    private final String topic;
    private final Boolean isAsync;

    /**
     * 构建方法，初始化生产者对象
     * @param topic
     * @param isAsync
     */
    public Producer(String topic, Boolean isAsync) {
        Properties props = new Properties();
        // 用户拉取kafka的元数据
        props.put("bootstrap.servers", "localhost:9092");
        props.put("client.id", "DemoProducer");
        //设置序列化的类。
        //二进制的格式
        props.put("key.serializer",
"org.apache.kafka.common.serialization.IntegerSerializer");
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        //消费者，消费数据的时候，就需要进行反序列化。
        // 初始化kafkaProducer
        producer = new KafkaProducer<>(props);
        this.topic = topic;
        this.isAsync = isAsync;
    }

    public void run() {
```

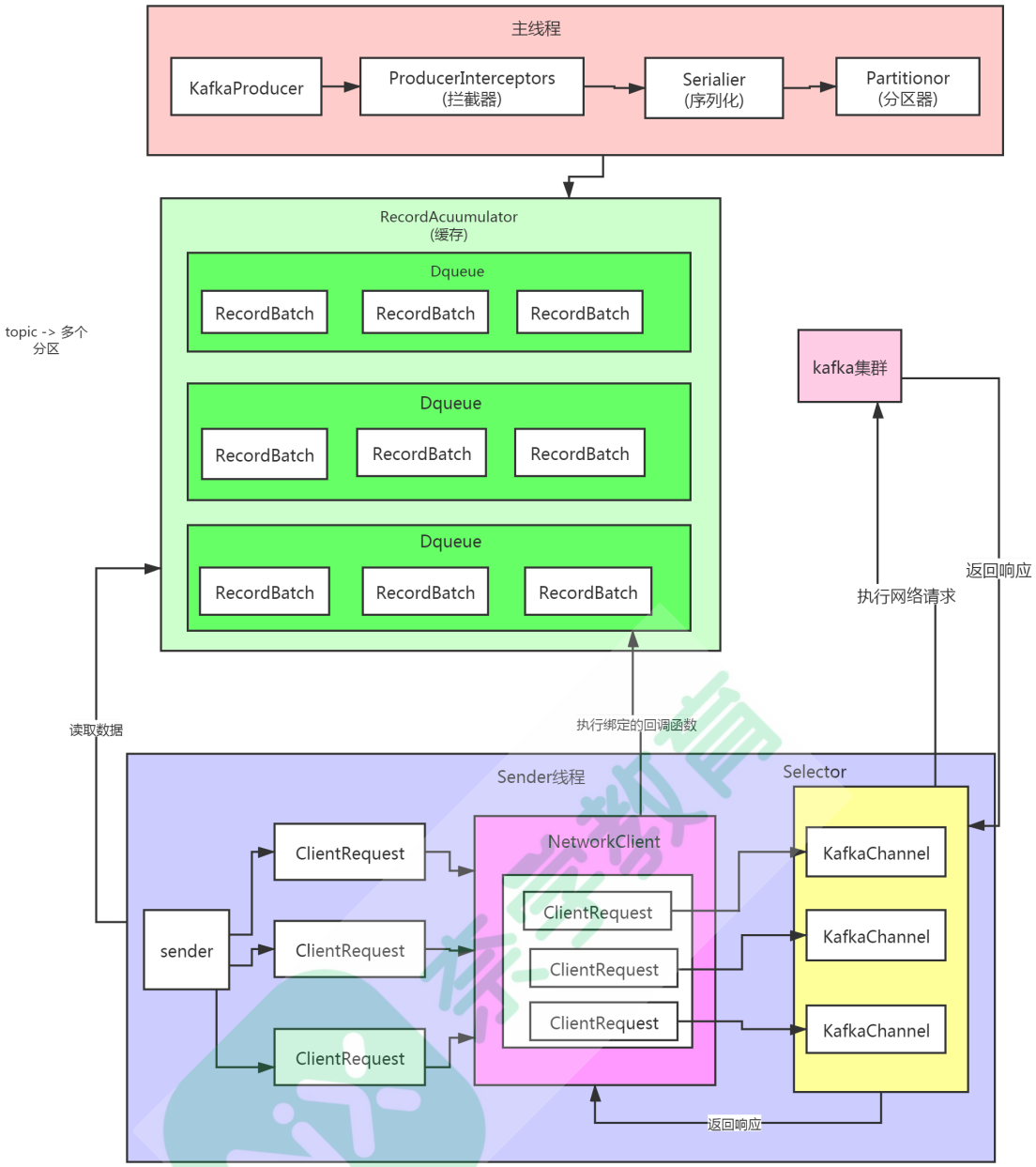
```
        int messageNo = 1;
        // 一直会往kafka发送数据
        while (true) {
            String messageStr = "Message_" + messageNo;
            long startTime = System.currentTimeMillis();
            //isAsync , kafka发送数据的时候，有两种方式
            //1: 异步发送
            //2: 同步发送
            //isAsync: true的时候是异步发送，false就是同步发送
            if (isAsync) { // Send asynchronously
                //异步发送
                //这样的方式，性能比较好，我们生产代码用的就是这种方式。
                producer.send(new ProducerRecord<>(topic,
                    messageNo,
                    messageStr), new DemoCallBack(startTime, messageNo,
messageStr));
            } else { // Send synchronously
                try {
                    //同步发送
                    //发送一条消息，等这条消息所有的后续工作都完成以后才继续下一条消息的发
送。
                    producer.send(new ProducerRecord<>(topic,
                        messageNo,
                        messageStr)).get();
                    System.out.println("Sent message: (" + messageNo + ", " +
messageStr + ")");
                } catch (InterruptedException | ExecutionException e) {
                    e.printStackTrace();
                }
            }
            ++messageNo;
        }
    }
}

class DemoCallBack implements Callback {

    private final long startTime;
    private final int key;
    private final String message;

    public DemoCallBack(long startTime, int key, String message) {
        this.startTime = startTime;
        this.key = key;
        this.message = message;
    }

    public void onCompletion(RecordMetadata metadata, Exception exception) {
        long elapsedTime = System.currentTimeMillis() - startTime;
        if(exception != null){
            System.out.println("有异常");
            //一般我们生产里面 还会有其它的备用的链路。
        }else{
            System.out.println("说明没有异常信息，成功的！！");
        }
        if (metadata != null) {
            System.out.println(
```

```
                "message(" + key + ", " + message + ") sent to partition(" +
metadata.partition() +
                    "), " +
                    "offset(" + metadata.offset() + ") in " + elapsedTime + "
ms");
        } else {
            exception.printStackTrace();
        }
    }
}
```

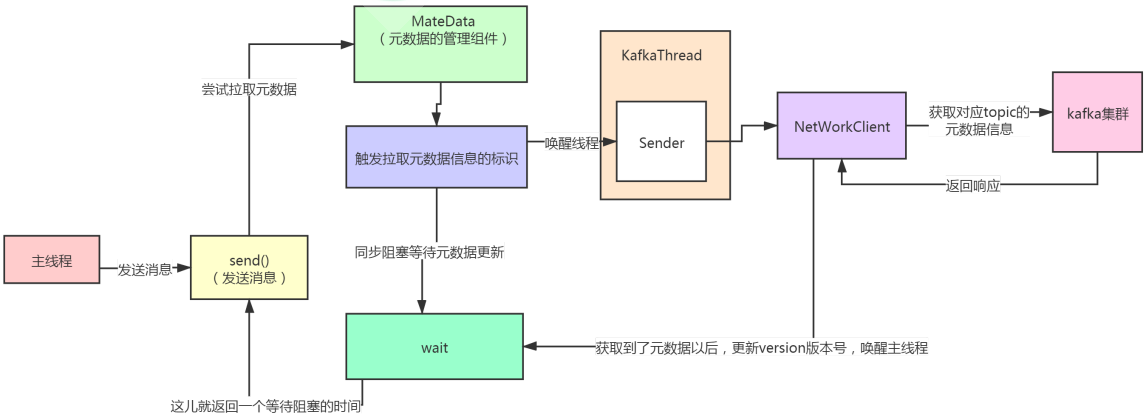
6.3 元数据信息关系



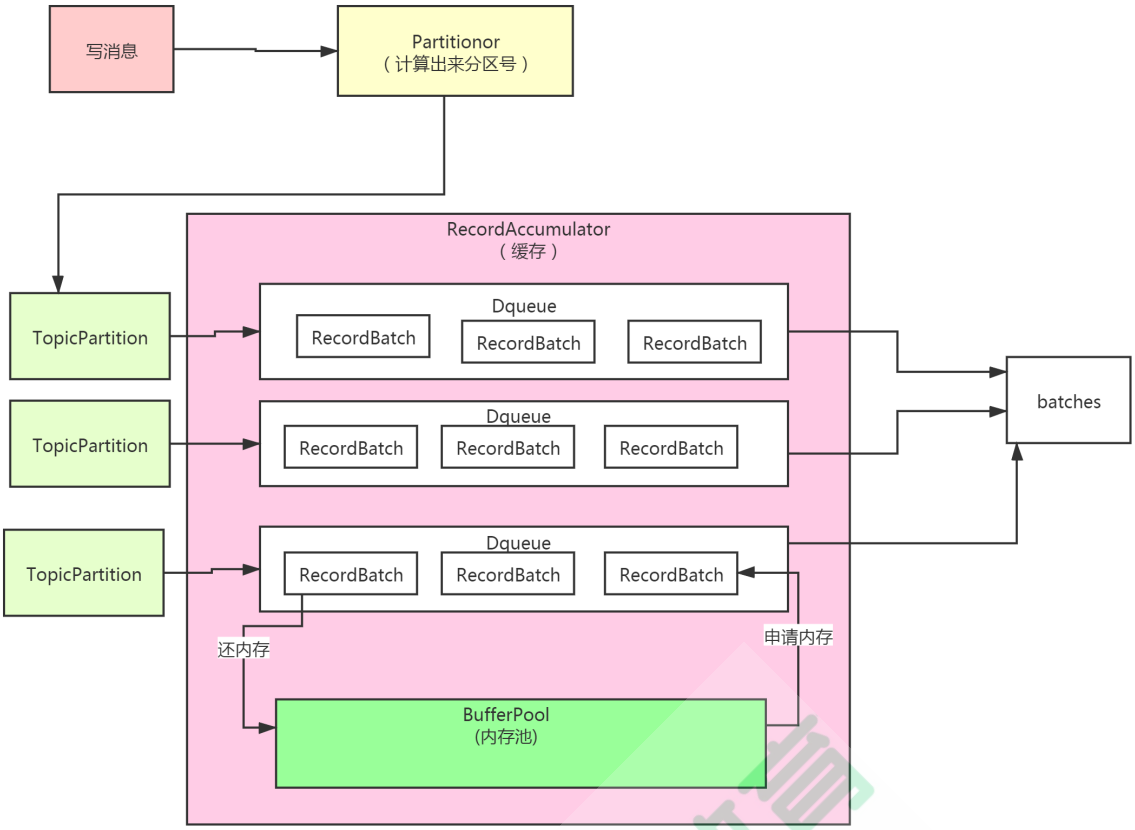
6.4 Producer核心流程深度剖析剖析



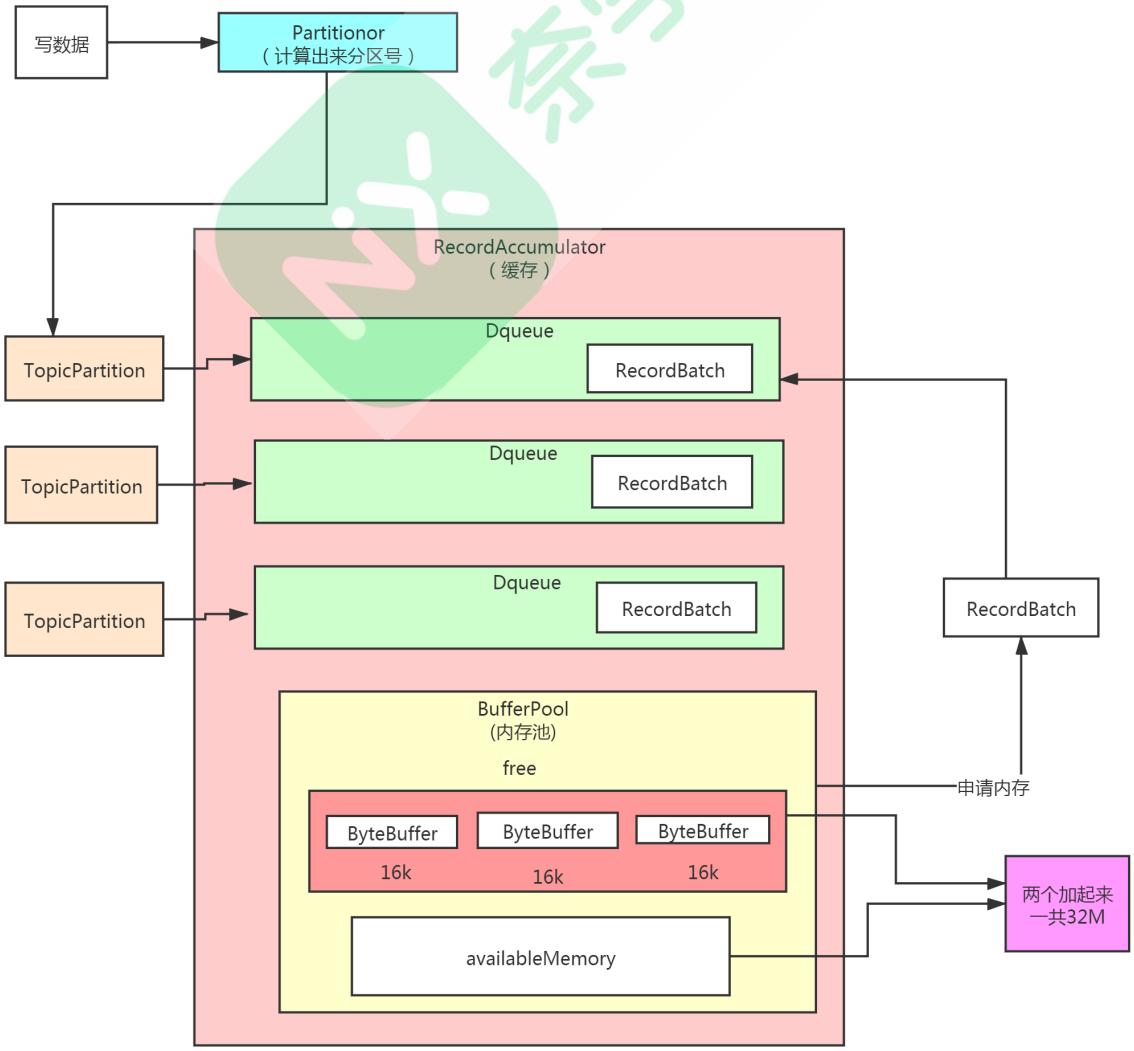
6.5 元数据加载流程剖析



6.6 RecordAccumulator原理


写消息
Partitionor
（计算出来分区号）
RecordAccumulator
（缓存）
TopicPartition
TopicPartition
TopicPartition
Dqueue
RecordBatch
RecordBatch
RecordBatch
Dqueue
RecordBatch
RecordBatch
RecordBatch
Dqueue
RecordBatch
RecordBatch
RecordBatch
batches
还内存
申请内存
BufferPool
(内存池)

6.7 内存池设计原理


写数据
Partitionor
（计算出来分区号）
RecordAccumulator
（缓存）
TopicPartition
TopicPartition
TopicPartition
Dqueue
RecordBatch
Dqueue
RecordBatch
Dqueue
RecordBatch
RecordBatch
BufferPool
(内存池)
free
ByteBuffer
ByteBuffer
ByteBuffer
16k
16k
16k
availableMemory
申请内存
两个加起来
一共32M

七 今日总结

01 源码阅读准备之基础知识准备

02 源码阅读准备之源码环境准备

03 源码阅读准备之源码剖析思路介绍

04 源码阅读准备之从一个demo入手

05 生产者源码之Producer核心流程介绍

06 生产者源码之Producer初始化

07 生产者源码之Producer端元数据管理

08 生产者源码之Producer源码核心流程初探

09 生产者源码之Producer加载元数据

10 生产者源码之分区选择

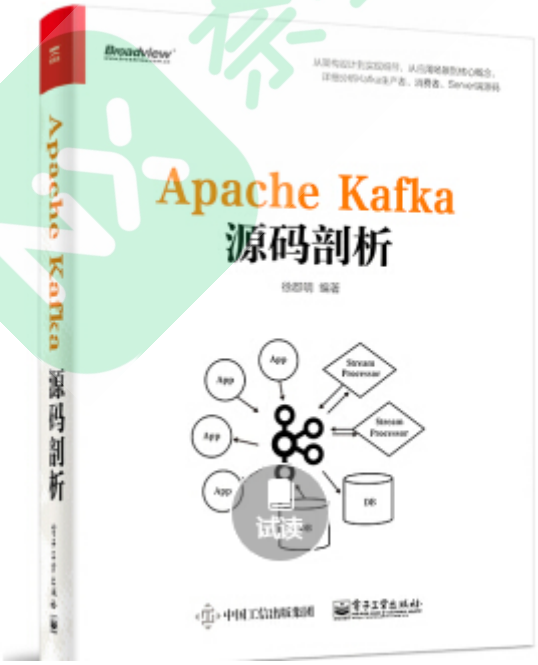
11 生产者源码之RecordAccumulator封装消息程初探

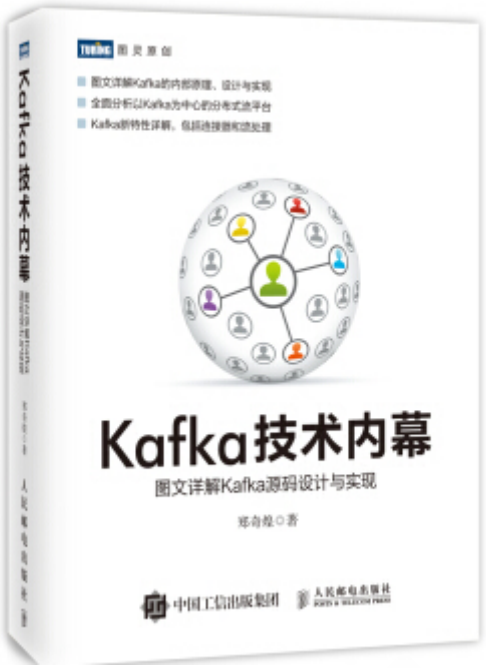
12 生产者源码之CopyOnWriteMap数据结构使用

13 生产者源码之把数据写入对应批次(分段加锁)

14 生产者源码之内存池设计

八 书籍推荐




图灵原创
图文详解Kafka的内部原理、设计与实现
全面分析以Kafka为中心的分布式消息平台
Kafka新特性详解，包括连接器和流处理
Kafka技术内幕
图文详解Kafka源码设计与实现
郑奇煌 著
中国工信出版集团
人民邮电出版社